Facultad Informática Universidad Murcia

Towards a unified programming model for diverse computing architectures: Experiences using PHAST

Eduardo José Gómez Hernández
Bajo la dirección de:
José Manuel García Carrasco

3 Julio 2019

Objetivos

Facultad Informática Universidad Murcia

- High Performance Parallel Programming Models
- Performance Portability
- Caffe & PHAST

Índice

- Desde el inicio, buscamos romper las barreras del rendimiento.
- La ley de Moore estableció la forma de desarrollar procesadores.
- Incrementar la frecuencia permitió aumentar el rendimiento de los circuitos.

CPU Limit

Facultad Informática Universidad Murcia

- Durante el inicio de los 2000, empezamos a encontrar un límite en la densidad energetica.
- Desarrollo de un numero paradigma, los multi-cores.
- Aún así, habían tareas que era mejor delegar a otros dispositivos.

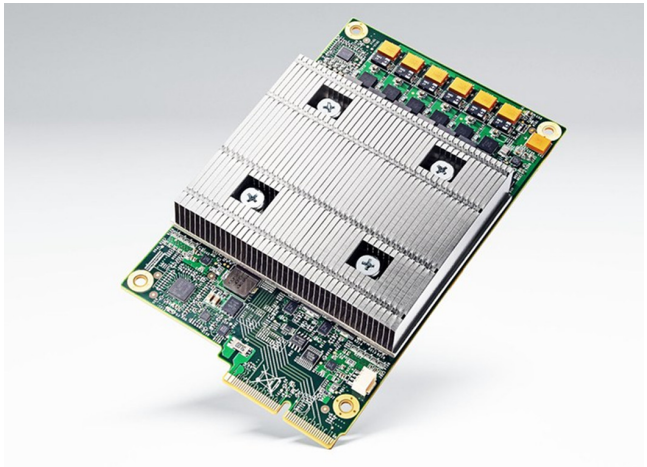
GPU

- Al principio fue usada únicamente para gráficos.
- Paralelismo SIMT (Single Instruction Multiple Threads).
- Muy extendido como acelerador, GPGPU.

FPGA & ASIC

- Las FPGAs eran usadas para crear prototipos de nuevo hardware.
- Ahora se empiezan a usar como hardware reconfigurable.
- Aún así, para tareas específicas los ASICs son mucho más eficientes que las FPGAs.

- Gran tiempo sin evolucionar por falta de poder de cómputo.
- Las redes neuronales es una de las técnicas más potentes.
- Está siendo usado en muchos campos, desde medicina hasta economía.
- Desarrollados usando frameworks optimizados debido a la cantidad de cómputo requerida.
- Reciente incremento del uso de aceleradores como las TPUs o los Tensor Cores, e incluso hardware más específico.



Performance Problems

- Los clústers tradicionales estaban sufriendo graves problemas para escalar ciertas aplicaciones.
- Un clúster heterogéneo permite una mejor escalabilidad, pero conlleva varios problemas.

- Uno de los problemas de los clúster heterogéneo es usarlo de manera eficiente.
- Existe una gran variedad de modelos para usar un clúster heterogéneo de manera eficiente.

Clasificación

- De 50 modelos revisados, hemos clasificado 23.
- Hemos basado la clasificación en 4 factores.
  - Plataforma
  - Tipo de modelo
  - Paradigma de programación
  - Paradigma de memoria

Clasificación

			Programing Paradigm								Memory Paradigm					
			Kernel	Task	Directive	C++ Template	Skeleton	Wrapper	Threads		Hierarchical	Shared	Implicit	Explicit	PGAS	Distributed
CPU	Type	Language	OpenCL	Cilk			Lift				OpenCL	Cilk	Lift			
		Extension		Cilk AllScale	OpenMP OpenAcc OmpSs						OpenMP	Cilk OpenMP	AllScale	OmpSs OpenAcc		
		Library	SyCL Kokkos PacxxV2 MultiController	TBB StarPU	MPI	Phast Dash Thrust Boost.Compute	SkePU SkeCL	EasyCL	Pthreads		SkePU		Phast	PacxxV2 EasyCL Thrust Boost.Compute SkeCL Kokkos MultiController StarPU Pthreads	Dash	SyCL TBB MPI

			Programing Model								Memory Model					
			Kernel	Task	Directive	C++ Template	Skeleton	Wrapper	Threads		Hierarchical	Shared	Implicit	Explicit	PGAS	Distributed
GPU	Type	Language	OpenCL Cuda				Lift				OpenCL Cuda		Lift			
		Extension			OpenMP OpenAcc OmpSs						OpenMP	OpenMP		OmpSs OpenAcc		
		Library	SyCL Kokkos PacxxV2 MultiController	StarPU		Phast Dash Thrust Boost.Compute	SkePU SkeCL	EasyCL			SkePU		Phast	PacxxV2 EasyCL Thrust Boost.Compute SkeCL Kokkos MultiController StarPU	Dash	SyCL

			Programing Model								Memory Model					
			Kernel	Task	Directive	C++ Template	Skeleton	Wrapper	Threads		Hierarchical	Shared	Implicit	Explicit	PGAS	Distributed
FPGA	Type	Language	OpenCL								OpenCL					
		Extension			OmpSs									OmpSs		
		Library	SyCL				SkeCL	EasyCL						EasyCL SkeCL		SyCL

Portable Models

- Modelos capaz de ser ejecutados en múltiples dispositivos usando el mismo código fuente.
- Ejemplos:
  - OpenMP (Algunas implementaciones)
  - OpenAcc
  - OpenCL & SYCL
  - SkePU
  - Kokkos
  - PHAST

Portable Models

- Modelos capaz de ser ejecutados en múltiples dispositivos usando el mismo código fuente.
- Ejemplos:
  - OpenMP (Algunas implementaciones)
  - OpenAcc
  - OpenCL & SYCL
  - SkePU
  - Kokkos
  - PHAST

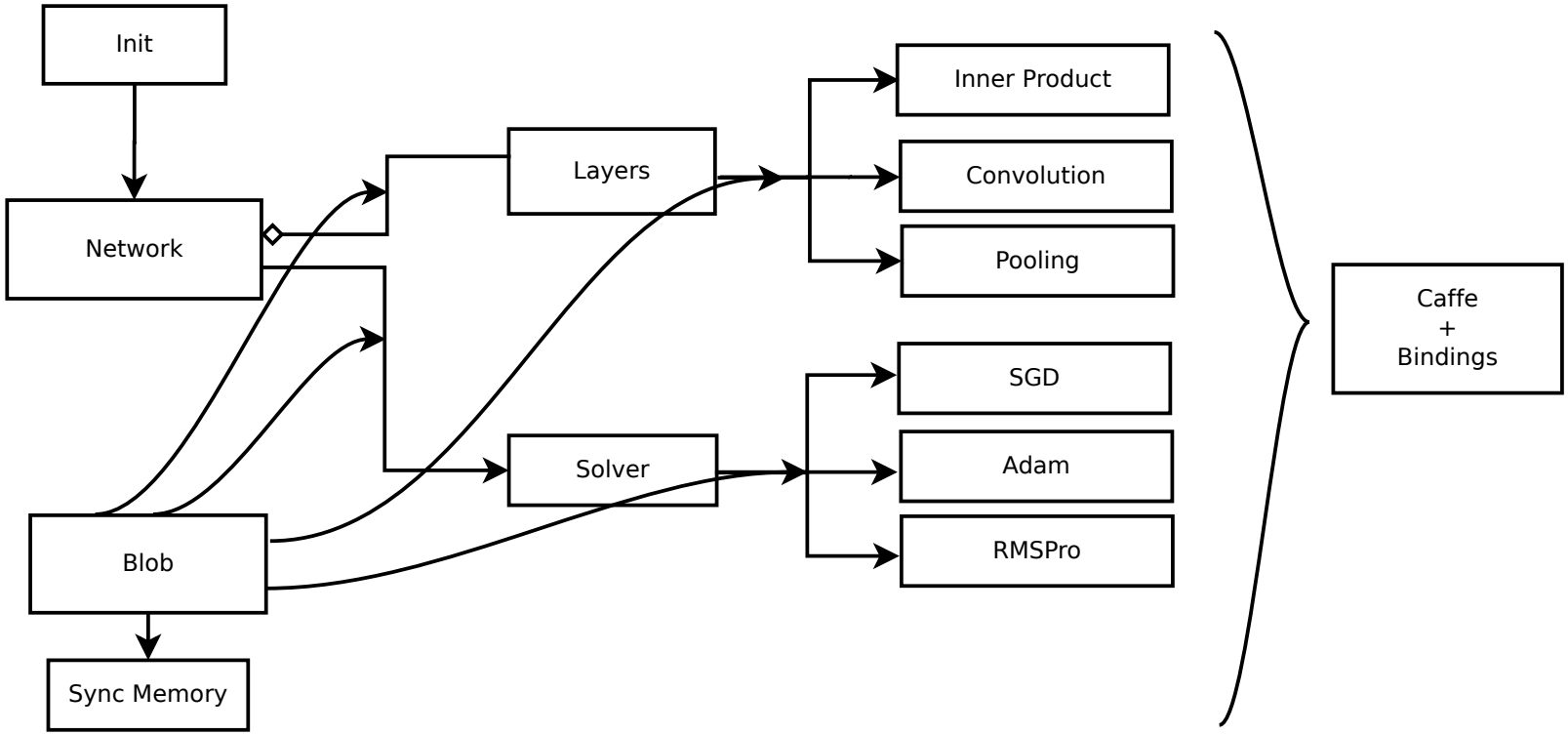
PHAST

- Librería de C++ con estructura similar a la librería algorítmica estándar de C++.
- Capaz de generar código para GPUs de Nvidia, y para multi- y many-core.
- Uso de `functors` para modificar los contenedores de datos.
- Define un modelo de programación agnóstico a la plataforma, pero permite el uso de porciones de código específico.

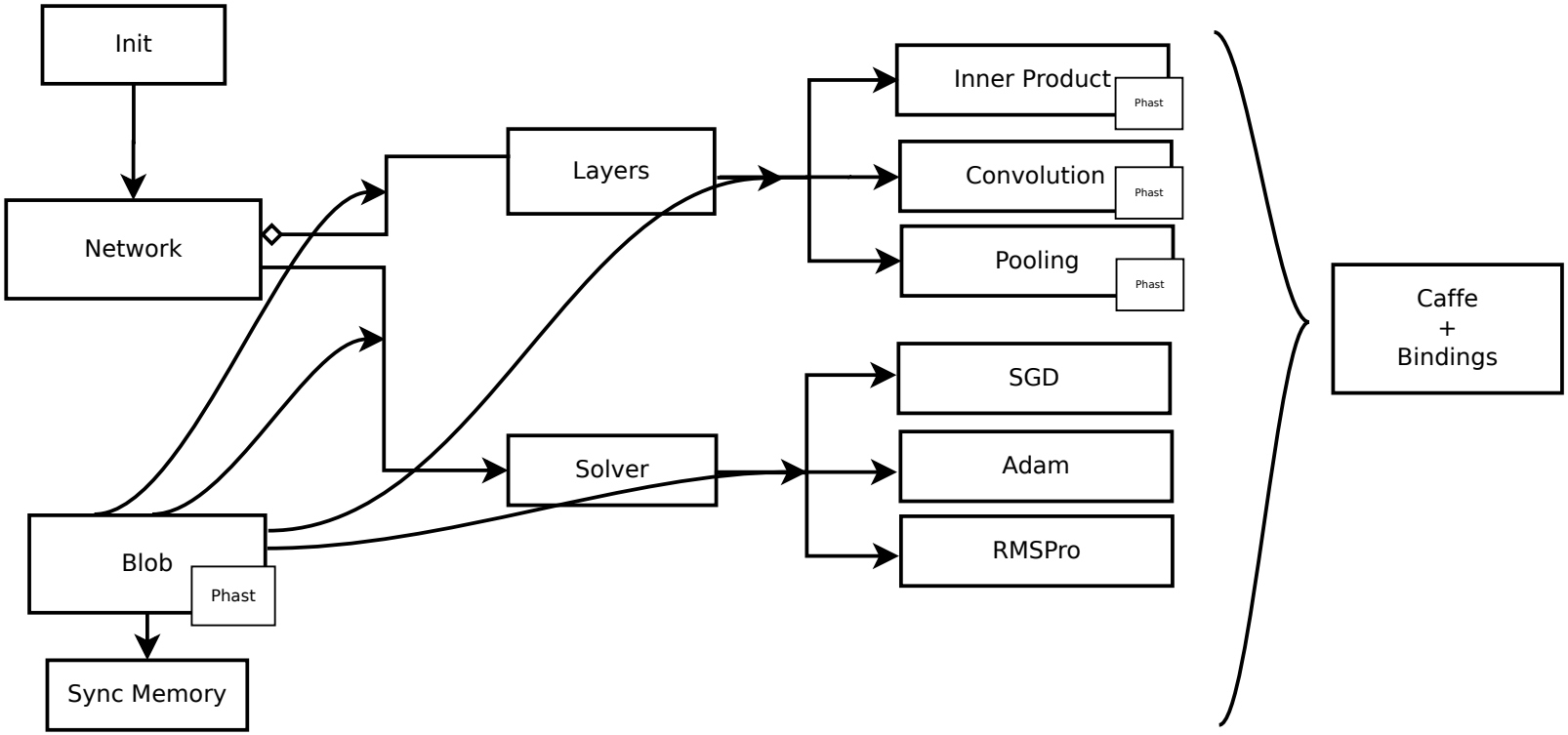
PHAST

Caffe

- Uno de los primeros frameworks de Deep Learning, desarrollado por Berkeley.
- Ha sido uno de los más usados hasta que terminó su desarrollo.
- Sigue siendo usado para investigación, gracias a su facilidad de ser modificado.



Roadmap



InnerProduct Caffe CPU

Facultad Informática Universidad Murcia

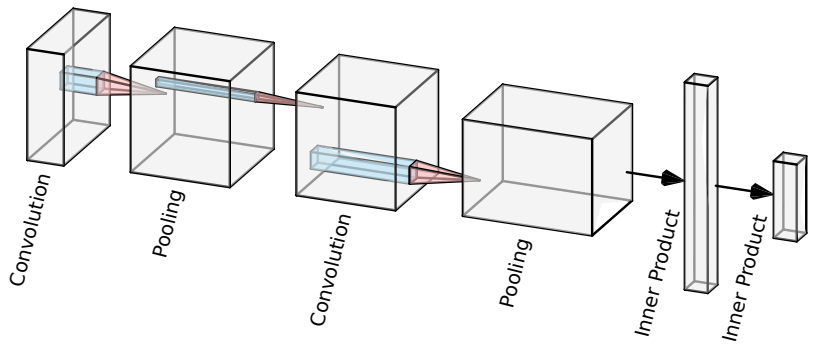

```
1  template <typename Dtype>
2  void InnerProductLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
3      const vector<Blob<Dtype>*>& top) {
4
5    const Dtype* bottom_data = bottom[0]->cpu_data();
6    Dtype* top_data = top[0]->mutable_cpu_data();
7    const Dtype* weight = this->blobs_[0]->cpu_data();
8
9    caffe_cpu_gemm<Dtype>(CblasNoTrans, transpose_ ? CblasNoTrans : CblasTrans,
10       M_, N_, K_, (Dtype)1.,
11       bottom_data, weight, (Dtype)0., top_data);
12
13   if (bias_term_) {
14     caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasNoTrans, M_, N_, 1, (Dtype)1.,
15         bias_multiplier_.cpu_data(),
16         this->blobs_[1]->cpu_data(), (Dtype)1., top_data);
17   }
18 }
```

InnerProduct PHAST


```
1  template <typename T, unsigned int policy = phast::get_default_policy()>
2  struct matrixPlusVectorRows : phast::functor::func_vec<T, policy> {
3    _PHAST_METHOD matrixPlusVectorRows() {}
4
5    _PHAST_METHOD void operator()(phast::functor::vector<T>& row) {
6      for(auto r = row.begin(), i = vec.begin(); r != row.end(); ++r, ++i)
7        *r += *i;
8    }
9    phast::functor::vector<T> vec;
10 };
11
12 template <>
13 void InnerProductLayer<float>::Forward_cpu(const vector<Blob<float>*>& bottom,
14   const vector<Blob<float>*>& top) {
15   phast::matrix<float> matA = bottom[0]->getDataAsMatrix(M_, K_, false);
16   phast::matrix<float> matB = this->blobs_[0]->getDataAsMatrix(K_, N_, !transpose_);
17   phast::matrix<float> matC = top[0]->getDataAsMatrix(M_, N_, false);
18
19   phast::dot_product(matA, matB, matC);
20
21   if (bias_term_) {
22     matrixPlusVectorRows<float> matrixPlusVectorRows;
23     matrixPlusVectorRows.vec.link(this->blobs_[1]->getDataAsVector(N_));
24     phast::for_each(matC.begin_i(), matC.end_i(), matrixPlusVectorRows);
25   }
26   if(!transpose_) matB.transpose();
27 }
```

TestBench

- Hemos usado dos redes neuronales basadas en LeNet para comprobar el funcionamiento de nuestras modificaciones.
- Hemos usado una máquina del clúster de GACOP, 2 Xeon CPU E5-2603 v3 y una GPU Nvidia GTX 1080.
- Ambas redes han sido ejecutadas correctamente en CPU y GPU.



- Caffe incluye una serie de tests para comprobar cada capa de manera individual.
- A excepción de la funcionalidad no implementada, todos los tests han pasado correctamente.

Performance

- Debido a la limitación de tiempo los resultados que tenemos, no son definitivos.
- Los casos de prueba son demasiado pequeños.
- El código puede ser mejorado fácilmente.

Conclusiones

Facultad Informática Universidad Murcia

- Clasificación de los modelos de programación más relevantes.
- Extracción de algunos modelos de programación portables.
- PHAST es factible para aplicaciones reales.

Trabajos Futuros

Facultad Informática Universidad Murcia

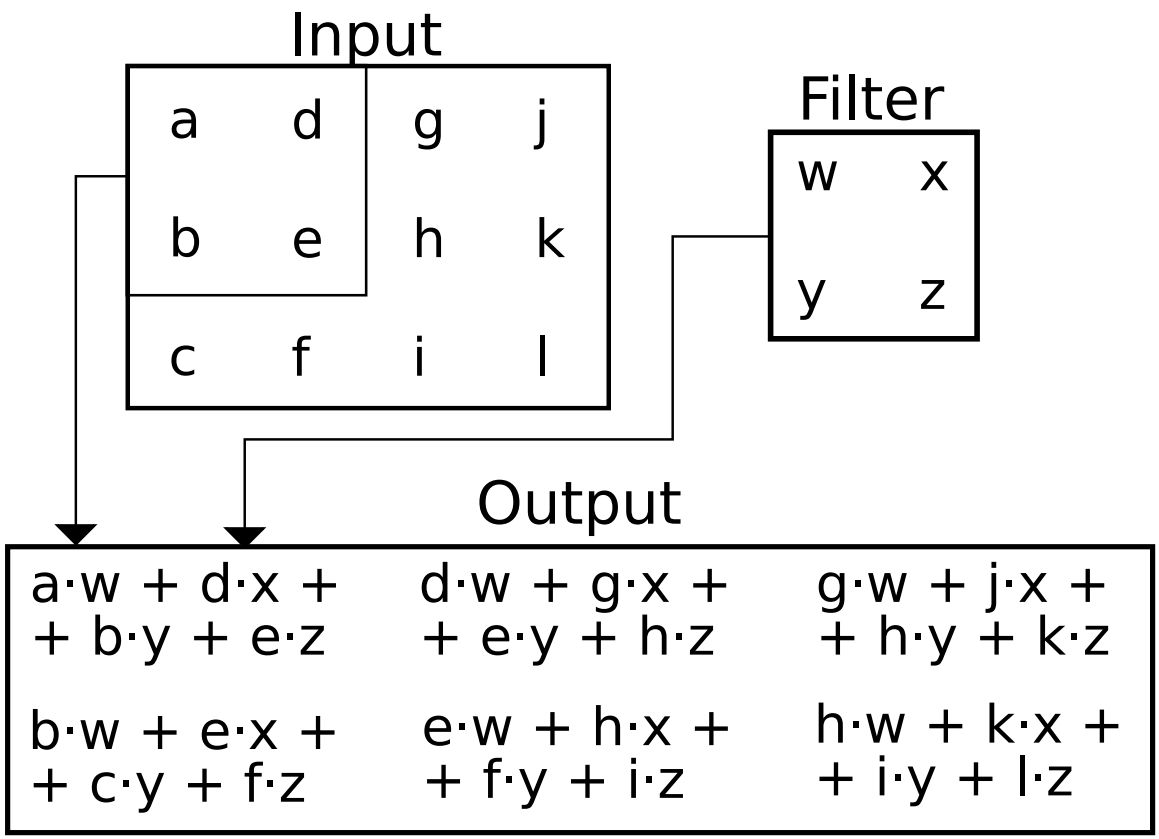
- Extender la clasificación de modelos.
- Completar y liberar la versión de Caffe con PHAST.
- Extender el trabajo a otros dispositivos.
- Mejorar el rendimiento.

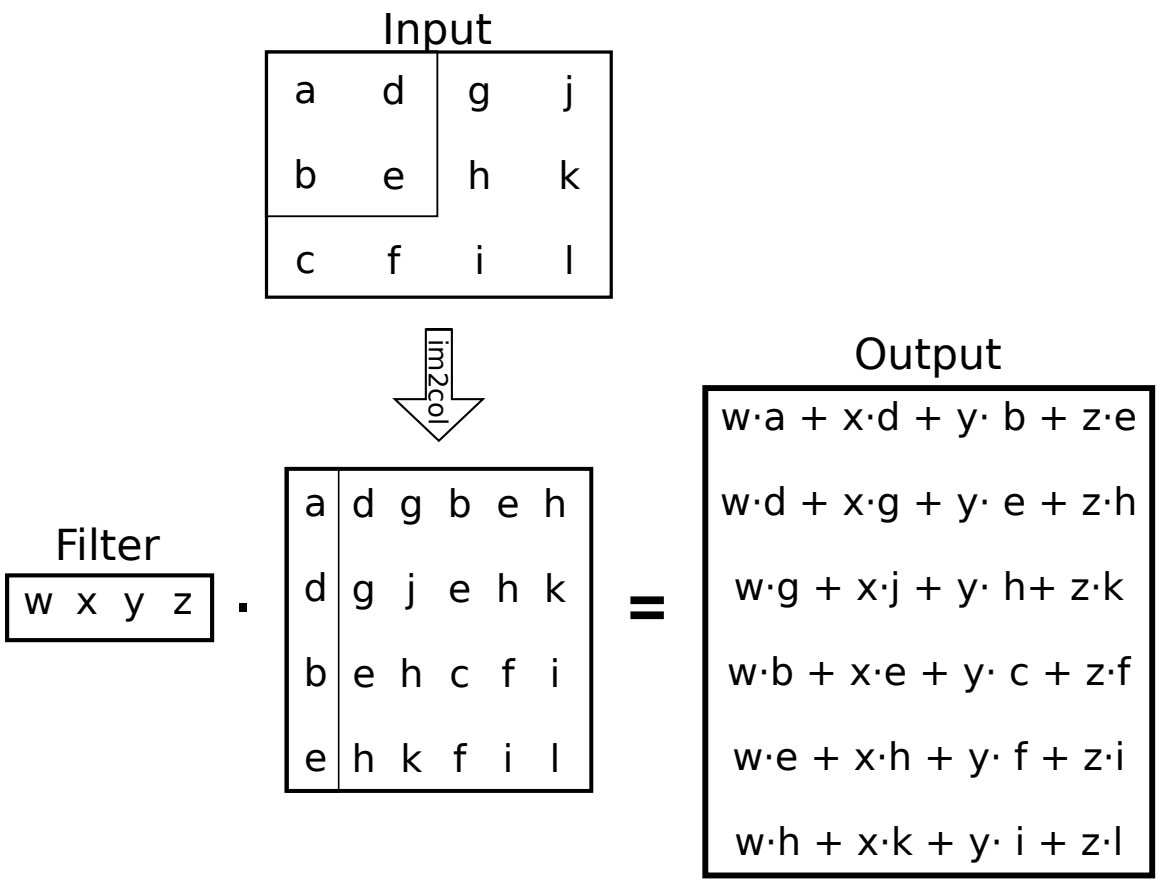
Facultad Informática Universidad Murcia

Towards a unified programming model for diverse computing architectures: Experiences using PHAST

Eduardo José Gómez Hernández
Bajo la dirección de:
José Manuel García Carrasco

3 Julio 2019





im2Col Caffe CPU

Facultad Informática Universidad Murcia


```
1   const int output_h = (height + 2 * pad_h - (dilation_h * (kernel_h - 1) + 1)) /
        stride_h + 1;
2   const int output_w = (width + 2 * pad_w - (dilation_w * (kernel_w - 1) + 1)) /
        stride_w + 1;
3   const int channel_size = height * width;
4   for (int channel = channels; channel--; data_im += channel_size) {
5   for (int kernel_row = 0; kernel_row < kernel_h; kernel_row++)
6       for (int kernel_col = 0; kernel_col < kernel_w; kernel_col++) {
7       int input_row = -pad_h + kernel_row * dilation_h;
8       for (int output_rows = output_h; output_rows; output_rows--) {
9           if (!is_a_ge_zero_and_a_lt_b(input_row, height)) {
10          for (int output_cols = output_w; output_cols; output_cols--)
11              *(data_col++) = 0;
12          } else {
13          int input_col = -pad_w + kernel_col * dilation_w;
14          for (int output_col = output_w; output_col; output_col--) {
15              if (is_a_ge_zero_and_a_lt_b(input_col, width)) {
16              *(data_col++) = data_im[input_row * width + input_col];
17              } else {
18              *(data_col++) = 0;
19              }
20              input_col += stride_w;
21          } }
22          input_row += stride_h;
23      } }
24  } }
```

im2Col PHAST


```
1   int index = this->get_index();
2
3   const int oih = (h + 2 * ph - (dh * (kh - 1) + 1)) / sh + 1;
4   const int oiw = (w + 2 * pw - (dw * (kw - 1) + 1)) / sw + 1;
5
6   const int irow = -ph + (((index / (oih*oiw)) % (kh * kw)) / kw) * dh + ((index % (oih*
        oiw)) / oiw) * sh;
7   const int icol = -pw + (((index / (oih*oiw)) % (kh * kw)) % kw) * dw + ((index % (oih*
        oiw)) % oiw) * sw;
8   if (irow >= 0 && irow < h && icol >= 0 && icol < w)
9     col = in.at(((index / (oih*oiw)) / (kh * kw)), irow * w + icol);
10  else col = 0;
```

Blob